

MyroC

2.4

Generated by Doxygen 1.8.8

Tue Aug 18 2015 11:20:50

Contents

1	Data Structure Index	1
1.1	Data Structures	1
2	File Index	3
2.1	File List	3
3	Data Structure Documentation	5
3.1	my_error_mgr Struct Reference	5
3.2	Picture Struct Reference	5
3.2.1	Detailed Description	6
3.2.2	Field Documentation	6
3.2.2.1	height	6
3.2.2.2	pix_array	6
3.2.2.3	width	6
3.3	Pixel Struct Reference	6
3.3.1	Detailed Description	6
4	File Documentation	7
4.1	MyroC.h File Reference	7
4.1.1	Detailed Description	9
4.1.2	Function Documentation	9
4.1.2.1	rBackward	9
4.1.2.2	rBeep	10
4.1.2.3	rBeep2	10
4.1.2.4	rConnect	11
4.1.2.5	rDisconnect	11
4.1.2.6	rDisplayPicture	11
4.1.2.7	rFastForward	12
4.1.2.8	rForward	12
4.1.2.9	rGetBattery	13
4.1.2.10	rGetBrightAll	13
4.1.2.11	rGetBrightTxt	14

4.1.2.12	rGetForwardness	15
4.1.2.13	rGetInfo	15
4.1.2.14	rGetIRAll	15
4.1.2.15	rGetIRTxt	16
4.1.2.16	rGetLightsAll	18
4.1.2.17	rGetLightTxt	19
4.1.2.18	rGetLine	20
4.1.2.19	rGetName	21
4.1.2.20	rGetObstacleAll	21
4.1.2.21	rGetObstacleTxt	22
4.1.2.22	rGetStall	23
4.1.2.23	rHardStop	24
4.1.2.24	rLoadPicture	24
4.1.2.25	rMotors	25
4.1.2.26	rSavePicture	25
4.1.2.27	rSetBluetoothEcho	26
4.1.2.28	rSetConnection	26
4.1.2.29	rSetForwardness	26
4.1.2.30	rSetForwardnessTxt	27
4.1.2.31	rSetIRPower	27
4.1.2.32	rSetLEDBack	27
4.1.2.33	rSetLEDFront	28
4.1.2.34	rSetName	28
4.1.2.35	rStop	28
4.1.2.36	rTakePicture	28
4.1.2.37	rTurnLeft	29
4.1.2.38	rTurnRight	29
4.1.2.39	rTurnSpeed	29

Chapter 1

Data Structure Index

1.1 Data Structures

Here are the data structures with brief descriptions:

my_error_mgr		5
Picture	Struct for a picture object	5
Pixel	Struct for a pixel	6

Chapter 2

File Index

2.1 File List

Here is a list of all documented files with brief descriptions:

MyroC.h	Header for a C-based, my-robot package for the Scribbler 2	7
-------------------------	--	---

Chapter 3

Data Structure Documentation

3.1 my_error_mgr Struct Reference

Data Fields

- struct jpeg_error_mgr **pub**
- jmp_buf **setjmp_buffer**

The documentation for this struct was generated from the following file:

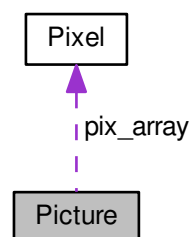
- MyroC.c

3.2 Picture Struct Reference

Struct for a picture object.

```
#include <MyroC.h>
```

Collaboration diagram for Picture:



Data Fields

- int `height`
- int `width`
- `Pixel pix_array` [192][256]

3.2.1 Detailed Description

Struct for a picture object.

Note

the picture size is always 256 in width and 192 in height
Following standard mathematical convention for a 2D matrix,
all references to a pixel are given within an array as [row][col]

3.2.2 Field Documentation

3.2.2.1 int height

The height of the image – set to 192 for robot camera

3.2.2.2 Pixel pix_array[192][256]

The array of pixels comprising the image

3.2.2.3 int width

The width of the image – set to 256 for robot camera

The documentation for this struct was generated from the following file:

- [MyroC.h](#)

3.3 Pixel Struct Reference

Struct for a pixel.

```
#include <MyroC.h>
```

Data Fields

- unsigned char [R](#)
The value of the red component.
- unsigned char [G](#)
The value of the green component.
- unsigned char [B](#)
The value of the blue component.

3.3.1 Detailed Description

Struct for a pixel.

The documentation for this struct was generated from the following file:

- [MyroC.h](#)

Chapter 4

File Documentation

4.1 MyroC.h File Reference

Header for a C-based, my-robot package for the Scribbler 2.

Data Structures

- struct [Pixel](#)
Struct for a pixel.
- struct [Picture](#)
Struct for a picture object.

Functions

- int [rConnect](#) (const char *address)
connects program to Scribbler
- void [rDisconnect](#) ()
disconnect program from Scribbler
- void [rSetConnection](#) (int new_socket_num)
set current connection to the socket number
- void [rBeep](#) (double duration, int frequency)
Beeps with the given duration and frequency.
- void [rBeep2](#) (double duration, int freq1, int freq2)
Generates two notes for the prescribed duration.
- void [rSetName](#) (const char *name)
Change name stored in the robot to the 16-byte name given.
- const char * [rGetName](#) ()
Get the name of the robot.
- void [rSetForwardness](#) (char *direction)
specifies which end of the Scribbler is considered the front
- void [rSetForwardnessTxt](#) (char *direction)
alternative to rSetForwardness for compatibility with earlier MyroC
- char * [rGetForwardness](#) ()
Gets the forwardness of the Scribbler.
- void [rSetLEDFront](#) (int led)
Set the front [fluke] LED on or off.
- void [rSetLEDBack](#) (double led)

- Set the the intensity of the back fluke LED,.*

 - double **rGetBattery** ()

Get the percentage of volts left in the batteries of the scribbler.
 - int **rGetStall** (int sampleSize)

Determine if robot has stalled Since readings of each brightness sensor can vary substantially, each sensor can be queried sampleSize times and an average obtained.
 - void **rSetBluetoothEcho** (char onOff)

Turn on and off echoing of Bluetooth transmissions All robot commands involve the transmission of a command over Bluetooth Scribbler commands are always 9 bytes Fluke commands have varying lengths The fluke echos most, but not all, of the commands For many commands, the fluke also echos 11 bytes of sensor data.
 - void **rGetLightsAll** (int lightSensors[3], int sampleSize)

Get the average values of each of the three light sensors in an array. Values of each light sensor can somewhat (typically under 5%-10%). To even out variability, the sensor can be queried sampleSize times and an average obtained.
 - int **rGetLightTxt** (const char *sensorName, int sampleSize)

Get the average values of a specified light sensor. Values of each light sensor can somewhat (typically under 5%-10%). To even out variability, the sensor can be queried sampleSize times and an average obtained.
 - void **rGetIRAll** (int irSensors[2], int sampleSize)

Get an array of true/false values regarding the presence of obstacle based on the average values of each of the three IR sensors. Since readings of each light sensor can vary substantially, each sensor can be queried sampleSize times and an average obtained.
 - int **rGetIRTxt** (const char *sensorName, int sampleSize)

Use specified IR sensor to determine if obstacle is present. Since values of each light sensor can vary substantially, the sensor can be queried sampleSize times and an average obtained.
 - void **rGetLine** (int lineSensors[2], int sampleSize)

Use Scribbler 2 line sensors of Scribbler to check for a black line on a white surface under the robot. Since values of each light sensor can vary substantially, the sensor can be queried sampleSize times and an average obtained.
 - void **rSetIRPower** (int power)

Set the amount of power for the dongle's IR sensors.
 - void **rGetObstacleAll** (int obstSensors[3], int sampleSize)

Get the average values of the three obstacle sensors in an array. Since readings of each obstacle sensor can vary substantially (successive readings may differ by several hundred or more), each sensor can be queried sampleSize times and an average obtained.
 - int **rGetObstacleTxt** (const char *sensorName, int sampleSize)

Get the average values of a specified obstacle (IR) sensor. Since values of each obstacle sensor can vary substantially (successive readings may differ by several hundred or more), the sensor can be queried sampleSize times and an average obtained.
 - void **rGetBrightAll** (int brightSensors[3], int sampleSize)

Read the Fluke's virtual light sensors. Since readings of each brightness sensor can vary substantially (successive readings may differ by 5000-10000), each sensor can be queried sampleSize times and an average obtained.
 - int **rGetBrightTxt** (char *sensorName, int sampleSize)

Reads one of the Fluke's virtual light sensors. Since values of each obstacle sensor can vary substantially (successive readings may differ by 5000-10000), the sensor can be queried sampleSize times and an average obtained.
 - void **rGetInfo** (char *infoBuffer)

returns information about the robot's dongle, firmware, and communication mode as a 60 character array in infoBuffer.
 - void **rTurnLeft** (double speed, double time)

turn Scribbler left for a specified time and speed
 - void **rTurnRight** (double speed, double time)

turn Scribbler right for a specified time and speed
 - void **rTurnSpeed** (char *direction, double speed, double time)

turn Scribbler in direction for a specified time and speed
 - void **rForward** (double speed, double time)

moves Scribbler forward for a specified time and speed
 - void **rFastForward** (double time)

- moves Scribbler forward at the largest possible speed for a specified time*
- void **rBackward** (double speed, double time)
- moves Scribbler backward for a specified time and speed*
- void **rMotors** (double leftSpeed, double rightSpeed)
- move robot with given speeds for the left and right motors continues until given another motion command or disconnected (non-blocking)*
- void **rStop** ()
- directs robot to stop movement*
- void **rHardStop** ()
- cuts power to the motor of the robot*
- **Picture** **rTakePicture** ()
- Use the camera to take a photo.*
- void **rSavePicture** (**Picture** pic, char *filename)
- Save a **Picture** to a .jpeg.*
- **Picture** **rLoadPicture** (char *filename)
- Load a picture from a .jpeg file.*
- void **rDisplayPicture** (**Picture** pic, double duration, const char *windowTitle)
- Display a picture in a new window.*

4.1.1 Detailed Description

Header for a C-based, my-robot package for the Scribbler 2.

Authors

Spencer Liberto
 Dilan Ustek
 Jordan Yuan
 Vasilisa Bashlovkina
 Anita DeWitt
 Jason Liu
 Nick Knoebber
 Henry M. Walker

Based on a C++ package by April O'Neill, David Cowden, Dilan Ustek, Erik Opavsky, and Henry M. Walker

Note

There is an r in the beginning of every function to make it easier to understand whether it is a robot function.

4.1.2 Function Documentation

4.1.2.1 void rBackward (double speed, double time)

moves Scribbler backward for a specified time and speed

Parameters

<i>speed</i>	the rate at which the robot should move backward linear range: -1.0 specifies move forward at full speed 0.0 specifies no forward/backward movement 1.0 specifies move backward at full speed
--------------	---

<i>time</i>	specifies the duration of the turn if negative: robot continues to go backward until given another motion command or disconnected (non-blocking) if nonnegative: robot moves backward for the given duration, in seconds
-------------	--

moves Scribbler backward for a specified time and speed

Parameters

<i>speed</i>	the rate at which the robot should move backward linear range: -1.0 specifies move forward at full speed 0.0 specifies no forward/backward movement 1.0 specifies move backward at full speed
<i>time</i>	specifies the duration of the turn if negative: robot continues to go backward until given another motion command or disconnected (non-blocking) if nonnegative: robot moves backward for the given duration, in seconds

4.1.2.2 void rBeep (double *duration*, int *frequency*)

Beeps with the given duration and frequency.

Parameters

<i>duration</i>	length of note in seconds
<i>frequency</i>	frequency of pitch in cycles per second (hertz)

Precondition

$duration > 0.0$

Beeps with the given duration and frequency

Parameters

<i>duration</i>	length of note in seconds
<i>frequency</i>	frequency of pitch in cycles per second (hertz)

Precondition

$duration > 0.0$

4.1.2.3 void rBeep2 (double *duration*, int *freq1*, int *freq2*)

Generates two notes for the prescribed duration.

Parameters

<i>duration</i>	length of note in seconds
<i>freq1</i>	frequency of first pitch in cycles per second (hertz)
<i>freq2</i>	frequency of second pitch in cycles per second (hertz)

Precondition

$duration > 0.0$

Generates two notes for the prescribed duration

Parameters

<i>duration</i>	length of note in seconds
<i>freq1</i>	frequency of first pitch in cycles per second (hertz)
<i>freq2</i>	frequency of second pitch in cycles per second (hertz)

Precondition

$duration > 0.0$

4.1.2.4 int rConnect (const char * *address*)

connects program to Scribbler

Parameters

<i>address</i>	string, giving name of workstation port or a Scribbler Bluetooth designation
----------------	--

several string formats are possible a communications port, such as "/dev/rfcomm0" a MAC address, such as "00:1E:19:01:0E:13" a Scribbler 2 fluke serial number, such as "245787" a full IPRE serial number, such as "IPRE245787" a Fluke 2 serial number (hexadecimal), such as "021F" a full Fluke 2 serial number, such as "Fluke2-021F"

Returns

the socket number of communications port

Postcondition

subsequent communications will take place through this socket, unless changed by rSetConnection

connects program to Scribbler

Parameters

<i>address</i>	string, giving name of workstation port or a Scribbler Bluetooth designation
----------------	--

several string formats are possible a communications port, such as "/dev/rfcomm0" a MAC address, such as "00:1E:19:01:0E:13" a Scribbler 2 fluke serial number, such as "245787" a full IPRE serial number, such as "IPRE245787" a Fluke 2 serial number (hexadecimal), such as "021F" a full Fluke 2 serial number, such as "Fluke2-021F"

Returns

the socket number of communications port

Postcondition

subsequent communications will take place through this socket, unless changed by rSetConnection

4.1.2.5 void rDisconnect ()

disconnect program from Scribbler

disconnect program from Scribbler

4.1.2.6 void rDisplayPicture (Picture *pic*, double *duration*, const char * *windowTitle*)

Display a picture in a new window.

Parameters

<i>pic</i>	RGB picture struct from Scribbler 2 camera
<i>duration</i>	if duration > 0, operation is blocking if duration <= 0, operation is non-blocking for duration != 0, picture displayed for abs(duration) seconds or until picture closed manually if duration == 0, picture displayed until closed manually
<i>windowTitle</i>	The title of the window that appears. white spaces will be replaced with underscores.

Precondition

windowTitle is less than 100 characters.

Display a picture in a new window

Parameters

<i>pic</i>	RGB picture struct from Scribbler 2 camera
<i>duration</i>	if duration > 0, operation is blocking if duration <= 0, operation is non-blocking for duration != 0, picture displayed for abs(duration) seconds or until picture closed manually if duration == 0, picture displayed until closed manually
<i>windowTitle</i>	The title of the window that appears. white spaces will be replaced with underscores.

Precondition

windowTitle is less than 100 characters.

4.1.2.7 void rFastForward (double *time*)

moves Scribbler forward at the largest possible speed for a specified time

Parameters

<i>time</i>	specifies the duration of movement if negative: robot continues to move forward until given another motion command or disconnected (non-blocking) if nonnegative: robot moves forward for the given duration, in seconds
-------------	--

Warning

may take longer than usual to execute

moves Scribbler forward at the largest possible speed for a specified time

Parameters

<i>time</i>	specifies the duration of movement if negative: robot continues to move forward until given another motion command or disconnected (non-blocking) if nonnegative: robot moves forward for the given duration, in seconds
-------------	--

Warning

may take longer than usual to execute

4.1.2.8 void rForward (double *speed*, double *time*)

moves Scribbler forward for a specified time and speed

Parameters

<i>speed</i>	the rate at which the robot should move forward linear range: -1.0 specifies move backward at full speed 0.0 specifies no forward/backward movement 1.0 specifies move forward at full speed
<i>time</i>	specifies the duration of movement if negative: robot continues to move forward until given another motion command or disconnected (non-blocking) if nonnegative: robot moves forward for the given duration, in seconds

moves Scribbler forward for a specified time and speed

Parameters

<i>speed</i>	the rate at which the robot should move forward linear range: -1.0 specifies move backward at full speed 0.0 specifies no forward/backward movement 1.0 specifies move forward at full speed
<i>time</i>	specifies the duration of movement if negative: robot continues to move forward until given another motion command or disconnected (non-blocking) if nonnegative: robot moves forward for the given duration, in seconds

4.1.2.9 double rGetBattery ()

Get the percentage of volts left in the batteries of the scribbler.

Returns

percentage of battery voltage

Get the percentage of volts left in the batteries of the scribbler

Warning

may not return useful values

Returns

percentage of battery voltage

4.1.2.10 void rGetBrightAll (int *brightSensors*[3], int *sampleSize*)

Read the Fluke's virtual light sensors. Since readings of each brightness sensor can vary substantially (successive readings may differ by 5000-10000), each sensor can be queried *sampleSize* times and an average obtained.

Parameters

<i>brightSensors</i>	array to store intensity values
<i>sampleSize</i>	how many readings are taken for each sensor

Precondition

space already allocated for *brightSensors* array *sampleSize* > 0

Postcondition

brightSensors[0] gives average value for left sensor *brightSensors*[1] gives average value for middle sensor *brightSensors*[2] gives average value for right sensor Brightness values near 0 represent bright light Brightness values may extend to about 65535 for a very dark region.

Read the Fluke's virtual light sensors. Since readings of each brightness sensor can vary substantially (successive readings may differ by 5000-10000), each sensor can be queried *sampleSize* times and an average obtained.

Parameters

<i>brightSensors</i>	array to store intensity values
<i>sampleSize</i>	how many readings are taken for each sensor

Precondition

space already allocated for brightSensors array $sampleSize > 0$

Postcondition

brightSensors[0] gives average value for left sensor brightSensors[1] gives average value for middle sensor
brightSensors[2] gives average value for right sensor Brightness values near 0 represent bright light Brightness
values may extend to about 65535 for a very dark region.

4.1.2.11 int rGetBrightTxt (char * *sensorName*, int *sampleSize*)

Reads one of the Fluke's virtual light sensors. Since values of each obstacle sensor can vary substantially (successive readings may differ by 5000-10000), the sensor can be queried *sampleSize* times and an average obtained.

Parameters

<i>sensorName</i>	name of the obstacle sensor
-------------------	-----------------------------

Precondition

sensorName is "left", "center", "middle", or "right" (not case sensitive) designations "center" and "middle" are alternatives for the same light sensor

Parameters

<i>sampleSize</i>	how many readings are taken for the sensor
-------------------	--

Precondition

$sampleSize > 0$

Returns

reading from the specified obstacle sensor, averaged over *sampleSize* number of data samples Brightness
values near 0 represent bright light Brightness values may extend to about 65535 for a very dark region.

Reads one of the Fluke's virtual light sensors. Since values of each obstacle sensor can vary substantially (successive readings may differ by 5000-10000), the sensor can be queried *sampleSize* times and an average obtained.

Parameters

<i>sensorName</i>	name of the obstacle sensor
-------------------	-----------------------------

Precondition

sensorName is "left", "center", "middle", or "right" (not case sensitive) designations "center" and "middle" are alternatives for the same light sensor

Parameters

<i>sampleSize</i>	how many readings are taken for the sensor
-------------------	--

Precondition

sampleSize > 0

Returns

reading from the specified obstacle sensor, averaged over *sampleSize* number of data samples Brightness values near 0 represent bright light Brightness values may extend to about 65535 for a very dark region.

4.1.2.12 char* rGetForwardness ()

Gets the forwardness of the Scribbler.

Returns

either "fluke-forward" or "scribbler-forward"

Gets the forwardness of the Scribbler

Returns

either "fluke-forward" or "scribbler-forward"

4.1.2.13 void rGetInfo (char * infoBuffer)

returns information about the robot's dongle, firmware, and communication mode as a 60 character array in info↵ Buffer.

Parameters

<i>infoBuffer</i>	a pre-defined, 60-character array
-------------------	-----------------------------------

Postcondition

infoBuffer contains relevant robot information

returns information about the robot's dongle, firmware, and communication mode as a 60 character array in info↵ Buffer.

Parameters

<i>infoBuffer</i>	a pre-defined, 60-character array
-------------------	-----------------------------------

Postcondition

infoBuffer contains relevant robot information

4.1.2.14 void rGetIRall (int irSensors[2], int sampleSize)

Get an array of true/false values regarding the presence of obstacle based on the average values of each of the three IR sensors. Since readings of each light sensor can vary substantially, each sensor can be queried sample↵ Size times and an average obtained.

Parameters

<i>irSensors</i>	array to store intensity values
<i>sampleSize</i>	how many readings are taken for each sensor

Precondition

space already allocated for irSensors array $sampleSize > 0$

Postcondition

irSensors[0] checks obstacle for left sensor irSensors[1] checks obstacle for right sensor
for each irSensors array value return 0 indicates no obstacle detected return 1 indicates obstacle detected

Get an array of true/false values regarding the presence of an obstacle, based on the average values of each of the three IR sensors. Since readings of each light sensor can vary substantially, each sensor can be queried *sampleSize* times and an average obtained.

Parameters

<i>irSensors</i>	array to store intensity values
<i>sampleSize</i>	how many readings are taken for each sensor

Precondition

space already allocated for irSensors array $sampleSize > 0$

Postcondition

irSensors[0] checks obstacle for left sensor irSensors[1] checks obstacle for right sensor
for each irSensors array value return 0 indicates no obstacle detected return 1 indicates obstacle detected

4.1.2.15 int rGetlRTxt (const char * *sensorName*, int *sampleSize*)

Use specified IR sensor to determine if obstacle is present. Since values of each light sensor can vary substantially, the sensor can be queried *sampleSize* times and an average obtained.

Parameters

<i>sensorName</i>	name of the light sensor
-------------------	--------------------------

Precondition

sensorName is "left" or "right" (not case sensitive)

Parameters

<i>sampleSize</i>	how many readings are taken for the sensor
-------------------	--

Precondition

$sampleSize > 0$

Returns

true/false (0/1) determination of obstacle, based on IR *sensorName* sensor, averaged over *sampleSize* number of data samples

Postcondition

return 0 indicates no obstacle detected return 1 indicates obstacle detected

Use specified IR sensor to determine if obstacle is present. Since values of each light sensor can vary substantially, the sensor can be queried sampleSize times and an average obtained.

Parameters

<i>sensorName</i>	name of the light sensor
-------------------	--------------------------

Precondition

sensorName is "left" or "right" (not case sensitive)

Parameters

<i>sampleSize</i>	how many readings are taken for the sensor
-------------------	--

Precondition

sampleSize > 0

Returns

true/false (0/1) determination of obstacle, based on IR sensorName sensor, averaged over sampleSize number of data samples

Postcondition

return 0 indicates no obstacle detected return 1 indicates obstacle detected

4.1.2.16 void rGetLightsAll (int *lightSensors*[3], int *sampleSize*)

Get the average values of each of the three light sensors in an array. Values of each light sensor can somewhat (typically under 5%-10%). To even out variability, the sensor can be queried sampleSize times and an average obtained.

Parameters

<i>lightSensors</i>	array to store intensity values
<i>sampleSize</i>	how many readings are taken for each sensor

Precondition

space already allocated for lightSensors array sampleSize > 0

Postcondition

lightSensors[0] gives average value for left sensor lightSensors[1] gives average value for middle sensor lightSensors[2] gives average value for right sensor Intensity values near 0 represent bright light Intensities may extend to about 65000 for a very dark region.

Get the average values of each of the three light sensors in an array. Values of each light sensor can somewhat (typically under 5%-10%). To even out variability, the sensor can be queried sampleSize times and an average obtained.

Parameters

<i>lightSensors</i>	array to store intensity values
---------------------	---------------------------------

<i>sampleSize</i>	how many readings are taken for each sensor
-------------------	---

Precondition

space already allocated for lightSensors array $sampleSize > 0$

Postcondition

lightSensors[0] gives average value for left sensor lightSensors[1] gives average value for middle sensor lightSensors[2] gives average value for right sensor Intensity values near 0 represent bright light Intensities may extend to about 65000 for a very dark region.

4.1.2.17 int rGetLightTxt (const char * *sensorName*, int *sampleSize*)

Get the average values of a specified light sensor. Values of each light sensor can somewhat (typically under 5%-10%). To even out variability, the sensor can be queried *sampleSize* times and an average obtained.

Parameters

<i>sensorName</i>	name of the light sensor
-------------------	--------------------------

Precondition

sensorName is "left", "center", "middle", or "right" (not case sensitive) designations "center" and "middle" are alternatives for the same light sensor

Parameters

<i>sampleSize</i>	how many readings are taken for the sensor
-------------------	--

Precondition

$sampleSize > 0$

Returns

reading from the specified light sensor, averaged over *sampleSize* number of data samples if *sensorName* invalid, returns -1.0

Get the average values of a specified light sensor. Values of each light sensor can vary somewhat (typically under 5%-10%). To even out variability, the sensor can be queried *sampleSize* times and an average obtained.

Parameters

<i>sensorName</i>	name of the light sensor
-------------------	--------------------------

Precondition

sensorName is "left", "center", "middle", or "right" (not case sensitive) designations "center" and "middle" are alternatives for the same light sensor

Parameters

<i>sampleSize</i>	how many readings are taken for the sensor
-------------------	--

Precondition

`sampleSize > 0`

Returns

reading from the specified light sensor, averaged over `sampleSize` number of data samples if `sensorName` invalid, returns -1.0

4.1.2.18 void rGetLine (int *lineSensors*[2], int *sampleSize*)

Use Scribbler 2 line sensors of Scribbler to check for a black line on a white surface under the robot. Since values of each light sensor can vary substantially, the sensor can be queried `sampleSize` times and an average obtained.

Warning

results of these sensors may be flakey!

Parameters

<i>lineSensors</i>	array to store line values detected
<i>sampleSize</i>	how many readings are taken for each sensor

Precondition

space already allocated for `lineSensors` array `sampleSize > 0`

Postcondition

`lineSensors[0]` checks left sensor for line `lineSensors[1]` checks right sensor for line
for each `irSensors` array value return 0 indicates line is identified return 1 indicates line is not identified

Use Scribbler 2 line sensors of Scribbler to check for a black line on a white surface under the robot. Since values of each light sensor can vary substantially, the sensor can be queried `sampleSize` times and an average obtained.

Warning

results of these sensors may be flakey!

Parameters

<i>lineSensors</i>	array to store line values detected
<i>sampleSize</i>	how many readings are taken for each sensor

Precondition

space already allocated for `lineSensors` array `sampleSize > 0`

Postcondition

`lineSensors[0]` checks left sensor for line `lineSensors[1]` checks right sensor for line
for each `irSensors` array value return 0 indicates line is identified return 1 indicates line is not identified

4.1.2.19 `const char* rGetName ( )`

Get the name of the robot.

Returns

information about the name of the robot

Postcondition

the returned name is a newly-allocated 17-byte string

4.1.2.20 `void rGetObstacleAll ( int obstSensors[3], int sampleSize )`

Get the average values of the three obstacle sensors in an array. Since readings of each obstacle sensor can vary substantially (successive readings may differ by several hundred or more), each sensor can be queried *sampleSize* times and an average obtained.

Parameters

<i>obstSensors</i>	array to store intensity values
<i>sampleSize</i>	how many readings are taken for each sensor

Precondition

space already allocated for *obstSensors* array *sampleSize* > 0

Postcondition

obstSensors[0] gives average value for left sensor *obstSensors*[1] gives average value for middle sensor *obstSensors*[2] gives average value for right sensor Obstacle values near 0 represent no obstacle is seen Obstacle values may approach 6000 as obstacle gets close.

Warning

As battery degrades, sensor readings degrade, yielding systematically lower numbers.

Get the average values of the three obstacle sensors in an array. Since readings of each obstacle sensor can vary substantially (successive readings may differ by several hundred or more), each sensor can be queried *sampleSize* times and an average obtained.

Parameters

<i>obstSensors</i>	array to store intensity values
<i>sampleSize</i>	how many readings are taken for each sensor

Precondition

space already allocated for *obstSensors* array *sampleSize* > 0

Postcondition

obstSensors[0] gives average value for left sensor *obstSensors*[1] gives average value for middle sensor *obstSensors*[2] gives average value for right sensor Obstacle values near 0 represent no obstacle is seen Obstacle values may approach 6000 as obstacle gets close.

Warning

As battery degrades, sensor readings degrade, yielding systematically lower numbers.

4.1.2.21 int rGetObstacleTxt (const char * *sensorName*, int *sampleSize*)

Get the average values of a specified obstacle (IR) sensor. Since values of each obstacle sensor can vary substantially (successive readings may differ by several hundred or more), the sensor can be queried *sampleSize* times and an average obtained.

Parameters

<i>sensorName</i>	name of the obstacle sensor
-------------------	-----------------------------

Precondition

sensorName is "left", "center", "middle", or "right" (not case sensitive) designations "center" and "middle" are alternatives for the same light sensor

Parameters

<i>sampleSize</i>	how many readings are taken for the sensor
-------------------	--

Precondition

space already allocated for vals array $\text{sampleSize} > 0$

Returns

reading from the specified obstacle sensor, averaged over *sampleSize* number of data samples Obstacle values near 0 represent no obstacle is seen Obstacle values may approach 6000 as obstacle gets close.

Warning

As battery degrades, sensor values degrade, yielding systematically lower numbers.

Get the average values of a specified obstacle (IR) sensor. Since values of each obstacle sensor can vary substantially (successive readings may differ by several hundred or more), the sensor can be queried *sampleSize* times and an average obtained.

Parameters

<i>sensorName</i>	name of the obstacle sensor
-------------------	-----------------------------

Precondition

sensorName is "left", "center", "middle", or "right" (not case sensitive) designations "center" and "middle" are alternatives for the same light sensor

Parameters

<i>sampleSize</i>	how many readings are taken for the sensor
-------------------	--

Precondition

space already allocated for vals array $\text{sampleSize} > 0$

Returns

reading from the specified obstacle sensor, averaged over *sampleSize* number of data samples Obstacle values near 0 represent no obstacle is seen Obstacle values may approach 6000 as obstacle gets close.

Warning

As battery degrades, sensor values degrade, yielding systematically lower numbers.

4.1.2.22 int rGetStall (int *sampleSize*)

Determine if robot has stalled Since readings of each brightness sensor can vary substantially, each sensor can be queried *sampleSize* times and an average obtained.

Parameters

<i>sampleSize</i>	how many readings are taken for each sensor
-------------------	---

Precondition

`sampleSize > 0`

Returns

whether or not robot current has stalled

Postcondition

Returns 1 if the robot has stalled Returns 0 otherwise.

Determine if robot has stalled Since readings of each brightness sensor can vary substantially, each sensor can be queried `sampleSize` times and an average obtained.

Parameters

<i>sampleSize</i>	how many readings are taken for each sensor
-------------------	---

Precondition

`sampleSize > 0`

Returns

whether or not robot current has stalled

Postcondition

Returns 1 if the robot has stalled Returns 0 otherwise.

4.1.2.23 void rHardStop ()

cuts power to the motor of the robot

cuts power to the motor of the robot

4.1.2.24 Picture rLoadPicture (char * filename)

Load a picture from a .jpeg file.

Parameters

<i>filename</i>	the name of the file
-----------------	----------------------

Precondition

file must exist

file must be a 256x192 .jpeg or .jpg

Returns

[Picture](#)

Load a picture from a .jpeg file.

Parameters

<i>filename</i>	the name of the file
-----------------	----------------------

Precondition

file must exist
file must be a 256x192 .jpeg or .jpg

Returns

[Picture](#)

4.1.2.25 void rMotors (double *leftSpeed*, double *rightSpeed*)

move robot with given speeds for the left and right motors continues until given another motion command or disconnected (non-blocking)

Parameters

<i>leftSpeed</i>	the rate at which the left wheel should turn linear range: -1.0 specifies move backward at full speed 0.0 specifies no forward/backward movement 1.0 specifies move forwardward at full speed
<i>rightSpeed</i>	the rate at which the right wheel should turn linear range: -1.0 specifies move backward at full speed 0.0 specifies no forward/backward movement 1.0 specifies move forward at full speed

move robot with given speeds for the left and right motors continues until given another motion command or disconnected (non-blocking)

Parameters

<i>leftSpeed</i>	the rate at which the left wheel should turn linear range: -1.0 specifies move backward at full speed 0.0 specifies no forward/backward movement 1.0 specifies move forwardward at full speed
<i>rightSpeed</i>	the rate at which the right wheel should turn linear range: -1.0 specifies move backward at full speed 0.0 specifies no forward/backward movement 1.0 specifies move forward at full speed

4.1.2.26 void rSavePicture ([Picture](#) *pic*, char * *filename*)

Save a [Picture](#) to a .jpeg.

Parameters

<i>pic</i>	RGB picture struct from Scribbler 2 camera
<i>filename</i>	the name of the file

Precondition

filename ends with .jpeg or .jpg.

Postcondition

If the file does not exist, a new file will be created.
If the file exists, the file will be overwritten.

Save a [Picture](#) to a .jpeg

Parameters

<i>pic</i>	RGB picture struct from Scribbler 2 camera
<i>filename</i>	the name of the file

Precondition

filename ends with .jpeg or .jpg.

Postcondition

If the file does not exist, a new file will be created.

If the file exists, the file will be overwritten.

4.1.2.27 void rSetBluetoothEcho (char *onOff*)

Turn on and off echoing of Bluetooth transmissions All robot commands involve the transmission of a command over Bluetooth Scribbler commands are always 9 bytes Fluke commands have varying lengths The fluke echos most, but not all, of the commands For many commands, the fluke also echos 11 bytes of sensor data.

Parameters

<i>onOff</i>	char 'y' enables echoing char 'n' disables echoing other character values ignored
--------------	---

4.1.2.28 void rSetConnection (int *new_socket_num*)

set current connection to the socket number

Parameters

<i>new_socket_↔ num</i>	the number of an open socket for communication
-----------------------------	--

Precondition

new_socket_num has been returned by rConnect the designated socket has not been closed

set current connection to the socket number

Parameters

<i>new_socket_↔ num</i>	the number of an open socket for communication
-----------------------------	--

Precondition

new_socket_num has been returned by rConnect the designated socket has not been closed

4.1.2.29 void rSetForwardness (char * *direction*)

specifies which end of the Scribbler is considered the front

Parameters

<i>direction</i>	identifies front direction
------------------	----------------------------

Precondition

direction is either "fluke-forward" or "scribbler-forward" (not case sensitive)

specifies which end of the Scribbler is considered the front

Parameters

<i>direction</i>	identifies front direction
------------------	----------------------------

Precondition

direction is either "fluke-forward" or "scribbler-forward" (not case sensitive)

4.1.2.30 void rSetForwardnessTxt (char * *direction*)

alternative to rSetForwardness for compatibility with earlier MyroC

alternative to rSetForwardness for compatibility with earlier MyroC

4.1.2.31 void rSetIRPower (int *power*)

Set the amount of power for the dongle's IR sensors.

Parameters

<i>power</i>	the desired power level for the IR sensors
--------------	--

Precondition

power is between 0 and 255 (inclusive)

Set the amount of power for the dongle's IR sensors

Parameters

<i>power</i>	the desired power level for the IR sensors
--------------	--

Precondition

power is between 0 and 255 (inclusive)

4.1.2.32 void rSetLEDBack (double *led*)

Set the the intensity of the back fluke LED,.

Parameters

<i>led</i>	intensity of the LED values between 0 and 1 provide a range of brightness from off to full intensity values bigger than 1 are treated as 1 (full brightness) values less than 0 are treated as 0 (LED off).
------------	---

Set the the intensity of the back fluke LED,

Parameters

<i>led</i>	intensity of the LED values between 0 and 1 provide a range of brightness from off to full intensity values bigger than 1 are treated as 1 (full brightness) values less than 0 are treated as 0 (LED off).
------------	---

4.1.2.33 void rSetLEDFront (int *led*)

Set the front [fluke] LED on or off.

Parameters

<i>led</i>	value 1 turns on LED value 0 turns off LED
------------	--

Precondition

led must be 0 or 1

Set the front [fluke] LED on or off

Parameters

<i>led</i>	value 1 turns on LED value 0 turns off LED
------------	--

Precondition

led must be 0 or 1

4.1.2.34 void rSetName (const char * *name*)

Change name stored in the robot to the 16-byte name given.

Parameters

<i>name</i>	specifies new name of robot if < 16 bytes given, name is filled with null characters if >= 16 bytes given, name is truncated to 15 bytes plus null
-------------	--

Change name stored in the robot to the 16-byte name given

Parameters

<i>name</i>	specifies new name of robot if < 16 bytes given, name is filled with null characters if >= 16 bytes given, name is truncated to 15 bytes plus null
-------------	--

4.1.2.35 void rStop ()

directs robot to stop movement

directs robot to stop movement

4.1.2.36 Picture rTakePicture ()

Use the camera to take a photo.

Returns

[Picture](#)

Use the camera to take a photo

Returns

[Picture](#)4.1.2.37 void rTurnLeft (double *speed*, double *time*)

turn Scribbler left for a specified time and speed

Parameters

<i>speed</i>	the rate at which the robot should move left linear range: -1.0 specifies right turn at full speed 0.0 specifies no turn 1.0 specifies left turn at full speed
<i>time</i>	specifies the duration of the turn if negative: robot continues to turn until given another motion command or disconnected (non-blocking) if nonnegative: robot turns for the given duration, in seconds

turn Scribbler left for a specified time and speed

Parameters

<i>speed</i>	the rate at which the robot should move left linear range: -1.0 specifies right turn at full speed 0.0 specifies no turn 1.0 specifies left turn at full speed
<i>time</i>	specifies the duration of the turn if negative: robot continues to turn until given another motion command or disconnected (non-blocking) if nonnegative: robot turns for the given duration, in seconds

4.1.2.38 void rTurnRight (double *speed*, double *time*)

turn Scribbler right for a specified time and speed

Parameters

<i>speed</i>	the rate at which the robot should move right linear range: -1.0 specifies left turn at full speed 0.0 specifies no turn 1.0 specifies right turn at full speed
<i>time</i>	specifies the duration of the turn if negative: robot continues to turn until given another motion command or disconnected (non-blocking) if nonnegative: robot turns for the given duration, in seconds

turn Scribbler right for a specified time and speed

Parameters

<i>speed</i>	the rate at which the robot should move right linear range: -1.0 specifies left turn at full speed 0.0 specifies no turn 1.0 specifies right turn at full speed
<i>time</i>	specifies the duration of the turn if negative: robot continues to turn until given another motion command or disconnected (non-blocking) if nonnegative: robot turns for the given duration, in seconds

4.1.2.39 void rTurnSpeed (char * *direction*, double *speed*, double *time*)

turn Scribbler in direction for a specified time and speed

Parameters

<i>direction</i>	direction of turn, based on looking from the center of the robot and facing forward
<i>speed</i>	the rate at which the robot should move forward linear range: -1.0 specifies turn at full speed 0.0 specifies no turn 1.0 specifies turn at full speed

<i>time</i>	specifies the duration of the turn if negative: robot continues to turn until given another motion command or disconnected if nonnegative: robot turns for the given duration, in seconds
-------------	---

Precondition

direction is "left" or "right", case insensitive

turn Scribbler in direction for a specified time and speed

Parameters

<i>direction</i>	direction of turn, based on looking from the center of the robot and facing forward
<i>speed</i>	the rate at which the robot should move forward linear range: -1.0 specifies turn at full speed 0.0 specifies no turn 1.0 specifies turn at full speed
<i>time</i>	specifies the duration of the turn if negative: robot continues to turn until given another motion command or disconnected if nonnegative: robot turns for the given duration, in seconds

Precondition

direction is "left" or "right", case insensitive

Index

height
 Picture, [6](#)

Picture, [5](#)
 height, [6](#)
 width, [6](#)

Pixel, [6](#)

width
 Picture, [6](#)